Engineer Guide: Device Changes

Register, compare, plan and apply device AAA configuration · Checked against tacctl 0.2.25

Contents

1. The change workflow at a glance 2
2. Registering devices 2
2.1 One device you know 2
2.2 Enroll an address (the usual way) 3
2.3 A blank Junos device reached with a local account 3
2.4 Keeping registry fields right 4
2.5 Replacing a device: a copy at a bench address 4
3. Reading and comparing 5
3.1 The AAA source addresses 6
4. Planning the change 6
4.1 The full walkthrough for a new device 7
4.2 Junos: the management filter 8
5. Applying the change 9
5.1 Junos: let tacctl apply it 9
5.2 What a shape is 11
5.3 Canary and confirmed 12
5.4 The options of a rollout 14
5.5 Cisco: paste the plan 14
5.6 WTI: type the checklist 15
6. Verify, then clean up 15
7. When it goes wrong 15
8. Quick reference 16

This guide is for engineers: members of a group at the **engineer** tier, who register network devices and bring their AAA configuration into line. Everything here applies to the devices of **your own scopes**. A device or address in another scope is refused.

It walks through a change from start to finish: register, read, compare, plan, apply or paste, verify, and back out if needed.

> **Example names used on this page** Scope: `site-a` · Devices: `sw1` (Cisco IOS-XE), `sw7` (IOS 12.4), `rtr1` (Junos), `jsw1` (blank Junos), `pdu1` (WTI) · You: `bob`

1. The change workflow at a glance

```
register ─► pull ─► diff / plan ─► apply (Junos) or paste (Cisco, WTI) ─► pull --diff ─►
done
                                          │
                                          └─► problem? ─► back-out (or superuser break-
glass)
```

Step	Command	Touches the device?
Register	`tacctl device add …` / `device enroll …`	Scans the host keys (and SNMP if set up)
Read	`tacctl device config pull <name>`	Logs in as you, **read only**
Compare	`tacctl device config diff <name>`	No (uses the last pull)
Plan	`tacctl device config plan <name>`	No (uses the last pull)
Apply (Junos)	`tacctl device config apply <name>`	Yes: `commit confirmed`, then proven
Verify	`tacctl device config pull <name> --diff`	Logs in as you, read only

---

2. Registering devices

2.1 One device you know

```
tacctl device add sw1 10.10.0.1 --vendor cisco --pull
tacctl device add rtr1 10.10.0.2 --vendor juniper --pull
tacctl device add sw7 10.20.0.7 --vendor cisco --legacy-ssh --ios-syntax ios12
tacctl device add pdu1 10.10.0.50 --vendor wti
```

- **Check the host key fingerprints** that `add` prints against the console: Cisco `show ip ssh`, Junos `file show /etc/ssh/ssh_host_ed25519_key.pub`. If you already have the fingerprint, `--host-key`

`SHA256:<fp>` pins only that key. A device that gives no key to the scan is read one key type at a time and then with tacctl's own ssh client, and a device that is slow to greet goes straight to that client and can take several seconds; when nothing works the refusal says what to do (*Connecting to Devices* §6).

- `--legacy-ssh` affects only the ssh algorithms (SHA-1 key exchange, CBC ciphers, ssh-rsa). The Cisco configuration syntax is set separately with `--ios-syntax auto|ios12|modern`. With `auto`, the version line of the pull decides.
- `--pull` reads the configuration straight after registering. A device that no scope covers is registered but not pulled.

2.2 Enroll an address (the usual way)

`device enroll` takes a new address from its host key to its plan in one attended run:

```
tacctl device enroll 10.10.0.5                # one new address
tacctl device enroll 10.10.0.21 10.10.0.22    # several, in order
tacctl device discover --snmp                 # unregistered addresses that log in,
                                              # with 'device add' lines
tacctl device enroll --discovered             # walk every one of them
```

For each address:

1. **Compare the key.** tacctl shows the one ssh key it will pin, with the console command that displays it (Cisco `show ip ssh`, Junos `file show /etc/ssh/ssh_host_ed25519_key.pub`). Compare them.
2. **Accept the key by typing the device's name.** Enter skips the device. Any other answer refuses it, with no second try. There is no `--yes` for the key: `--host-key SHA256:<fp>` (one address, the fingerprint you read on the console) answers it instead.
3. **Confirm the registration**, then give your password once for the pulls.
4. **Review the result.** tacctl reads the device, plans what needs changing, and for Junos offers the apply review. The table at the end gives each device's next command.

`enroll` refuses an address that is already registered, one that no scope covers, one outside your scopes, and one held for break-glass. A device stays registered if a later step fails.

2.3 A blank Junos device reached with a local account

```
# stop after the device's own check
tacctl device onboard 10.10.0.31 --login admin --name jsw1 --check
# full run
tacctl device onboard 10.10.0.31 --login admin --name jsw1
```

`onboard` pins the key you accept and checks the device (no pending commit, no AAA server already set). It registers, pulls and plans, then commits as the local account. Finally it proves the result with your own fresh TACACS+ login.

2.4 Keeping registry fields right

```
tacctl device description rtr1 "Internet edge A"
tacctl device location rtr1 --from-device   # SNMP sysLocation
tacctl device vendor sw7 --from-device      # vendor reported by SNMP
tacctl device sync --scope site-a           # preview: fill empty fields
                                            # from Junos pull records
tacctl device sync --scope site-a --apply
```

2.5 Replacing a device: a copy at a bench address

When a device is swapped for new hardware, register the new one as a copy of the old one at the address it has on the bench, set it up there, and move it into place:

```
# 1. register sw1-new as a copy of sw1, at its bench address
tacctl device clone sw1 sw1-new --staging 10.10.0.60
# 2. pin the new device's host key, pull, plan and apply as for any device
# 3. when it is in place: remove sw1 and give its name and address to sw1-new
tacctl device rename sw1-new sw1 --replace
```

- **What `clone` copies:** the registry settings, the SNMP settings and credentials, the NETCONF settings, the timing scale and the configuration blocks. **What it does not:** the address, the name, the pinned host keys, the acknowledged notices, a break-glass hold and the pull record, so pin its host keys and pull it as for any new device.
- `--staging` is required. The bench address is added to the scope of the original as a `/32` so the device can authenticate, or, when the scope answers it already, noted as staged. You can clone the devices of your own scopes, SNMP credentials included.
- **`rename --replace`** prints its plan lines, then asks one question (`--yes` is the only way to skip it; with no terminal it answers no). The original is removed with its files after a snapshot, the replacement takes its name and address, and the bench address ends.
- An original with a **break-glass hold** needs a superuser: the replace clears the hold (*Break Glass*), and an engineer is refused, naming it.
- A plain `device rename` (without `--replace`) and `host rename` carry the bench address that was staged under the old name.

---

3. Reading and comparing

```
tacctl device config pull sw1                       # one device
tacctl device config pull --scope site-a            # a whole scope
tacctl device config pull --vendor juniper --stale  # Junos: never/failed/differing/needs-
pull
tacctl device config list                           # state: ok / differs / never /
failed / pull
tacctl device config diff rtr1                      # section by section
```

How to read a diff:

Mark	Meaning
`-`	A statement tacctl renders that the device lacks
`+`	A statement in a managed hierarchy that tacctl does not render
`!`	A reordering
`?`	A secret your login could not see
`=`	A secret is present (its value is never compared or shown)

The managed sections are `aaa`, `roles`, `mgmt-acl`, `snmp`, `netconf`, and `breakglass`. Some scopes add a seventh, `system`.

> **Pull options:** `--concurrency <n>`, `--timeout <s>`, `--max-failures <n>` and `--transport auto|netconf|ssh`. A first Ctrl-C starts no new devices. A second one stops the devices that are running.

Where the time goes. For a run over many devices of which some are slow, add `--timing` to `device config pull`, `device config diff --pull`, `device config plan --pull` or `device config apply`. After the summary it prints, per device, the phases it went through in order with their time and its total, and names the slowest device:

```
  timing:
    sw1   connect 4.0s  login 900ms  session 1.2s  read 2.1s  compare 40ms  record 12ms
total 8.3s
    jsw1  connect 210ms  login 180ms  session 95ms  read 1.4s  compare 33ms  record 10ms
total 2.0s
  slowest: sw1 (8.3s)
```

The phases of a pull are `connect` (the TCP connect and the ssh key exchange up to the device's pinned host key), `login`, `session`, `read`, `compare` and `record`. An apply adds `preflight`, `load`, `check`, `commit`, `verify` (the fresh login and what it proves) and `confirm`; with `--pull`, the pull, the check and the apply of a device are added up. A phase the device did not reach is left out, not shown as zero, and a device that failed shows the phases it reached and, under the word of its status (`failed`, `auth-`

`failed`, `timeout`), the time it spent in the one it failed in. Nothing is stored, and without `--timing` the output is as it was.

- With `--json` each device's line has a `timings` object (the `steps` in order, each with its `ms`, and `total_ms`), and the summary line has `timings.slowest` and `timings.slowest_ms`.
- `diff --pull --json` prints the array of devices only: the slowest device's `timings` carries `"slowest": true`.
- `plan --pull` prints the timing on standard error, so standard output stays the plan.

A device that is slow for a reason of its own, such as a long round trip or a slow CPU, is a case for a timing scale, which a superuser sets (*Connecting to Devices* §7). You read the scale in force in `tacctl device show <name>` and `tacctl scope show <scope>`. A `--timeout` you type on a pull or an apply is the time of every device of that run, as typed, and wins over the scale.

3.1 The AAA source addresses

A device may send its AAA requests from an address other than the one it is registered at (a Junos `source-address`, an IOS `ip tacacs source-interface`). Two checks look at it:

- **The source must be an address of the device.** A source that no interface of the device holds is a difference: `aaa` reads `differs`, and the plan reports an `ACCESS` item that the paste never fixes (add the address to an interface, or change the source).
- **Does the address answer, and is it the same device?**

`bash tacctl device check sw1 --sources # one device (or --all) tacctl device scan --sources # after the scan, for every device`

Each source address read from the last pull is connected to and its ssh host key is compared with the one pinned for the device, without logging in. An address that does not answer, or that presents another key, is a warning (the address may belong to another device). Nothing is pinned or changed. An unanswered address costs about 8 seconds, and a server often has no path to a loopback, so unreachable may be by design.

The settings a device keeps for its own AAA server (a source address, a routing instance and the like) are no longer listed as differences.

---

4. Planning the change

```
tacctl device config plan sw1
# one file per device + COVER.txt
tacctl device config plan --scope site-a --out-dir ~/plans/site-a
# IOS: a restore point before the region
tacctl device config plan sw1 --restore-point=flash:
# the back-out region alone
tacctl device config plan rtr1 --backout
```

A plan has these parts:

1. **The report.** Each irregularity, with its severity (`ACCESS`, `MONITORING` or `COSMETIC`) and what the paste does about it, then two lines of information (below).
2. **A line to type first** that prints the device's name. Check it before you paste.
3. **One region** between `PASTE FROM HERE` and `PASTE TO HERE`, safe to paste whole.
4. **The check, the one line to type afterwards, and the back-out.**

Two lines of the report are information, not irregularities:

```
# Kept: snmp 2 statements of the device's own (communities, trap groups): left alone,
listed by
# device config diff.
# Secrets: 3 present (the TACACS+ server key, the SNMP community); only their presence is
checked,
# --secrets writes them again.
```

- **`Kept:`** counts, per section, the device's own statements that tacctl leaves alone: in `snmp` another community, trap groups or the `snmp name`; in `system` and `netconf` what the device holds beside tacctl's statements. The number is the one `device config list` shows as unmanaged, and the line never names a community or a value. A device whose only extras are of this kind reads `CONSISTENT: nothing to paste.`
- **`Secrets:`** names the kinds of secret present on the device, never a value. Secrets are compared by presence only, so the line stays the same once they are set; `--secrets` types them again. Secrets the pull's login could not see are counted apart as not visible.

Other statements outside the standard (*extras*: an old server, a class or a filter tacctl does not render) are reported as items and left alone.

`single-connection` is gone (0.2.24). tacquito closes a connection that has been idle for 15 seconds, and a later login over the dead socket failed, so the walkthroughs (`config cisco`, `config juniper`) no longer print `single-connection`. On a device that has it, `plan` and `apply` delete it from tacctl's server and `diff` and `plan` list it as an extra. The plan has a new shape, so `apply` sends its first device alone, and `apply --confirmed` takes it only after one device of it has been applied and confirmed again (§5.3). A Junos `single-connection` that only an apply-group gives is left alone (`inherited from a group: remove it from the group by hand`); another server's line and a custom template keep theirs. The upgrade changes no device: run `plan` and `apply` on the devices that have the line. `--converge` adds the cleanup **only after** the first paste has been verified. That way the paste that adds the new server is never the same paste that removes the old one.

A plan is refused (exit 1) if the device has never been pulled, its last pull is older than `stale-days`, its last pull failed, or its Cisco syntax is unknown. Pull again first, or use `--pull`.

4.1 The full walkthrough for a new device

To configure a device by hand from scratch, print its walkthrough:

```
# the device's data, then the walkthrough for its vendor and scope
tacctl device config show sw1
```

The output contains the scope's shared secret. Handle it as secret. Each print writes a `secret-read` line to the audit log.

The walkthrough tells the device the server address recorded with `tacctl server addresses` (else the detected one; `--server` wins), and says which address plays which role when that differs from the address tacctl reaches the device from. It follows the configured TACACS+ listener's port too: nothing is written for the default 49, and a listener on another port adds `port <n>` under `tacacs server TACACS` (Cisco), `set system tacplus-server <address> port <n>` (Junos) and the Authentication Port item (WTI). `diff` and `plan` render the same, so a device that still has the old port differs and its plan types the new one. A template you changed in an earlier release has none of the new variables: the upgrade leaves it and writes the new default beside it as `.new`.

4.2 Junos: the management filter

The Junos management filter is planned whole, in the mode of the device's scope: `ssh-only` (the default) or `strict`, deny by default (*Vendor Notes* §3.4). You can read the mode of your own scopes; the permit list, the filter names and setting the mode are the superuser's.

```
# the mode of the scope's filter and where it comes from
tacctl scope mgmt-acl site-a mode
# the address you are logged in from, for a device that applies the filter to lo0
tacctl device config plan rtr1 --from 192.0.2.10
```

When the mode changes, the next plan moves the device's filter to it, in either direction, and its report warns of what that does to the traffic:

- **To `strict`:** the traffic that stops once the filter is applied to `lo0` (routing protocols carried over IP, BFD among them, from neighbours that are not in the permit list, TACACS+ and RADIUS replies from servers that are not in it, NTP, syslog and DHCP, SNMP from sources outside it other than its clients, which still poll, every other protocol; IS-IS is not carried over IP, so the filter neither stops nor permits it), the sources the filter accepts, apart from the address it accepts the authentication server's answers from (`permit-auth-replies`, *Vendor Notes* §3.4), and to keep a console session open.
- **To `ssh-only`**, over a filter with no `default-accept` term: that the plan adds the term, so the device accepts everything again except ssh, NETCONF and SNMP from outside the list, and the command that sets `strict` instead.

Do not lock yourself out. When the device already applies the filter to `lo0`, a plan that changes the filter needs `--from <address>`, the address you are logged in from. Without it the report warns. With it, the filter's section is refused if the new filter would not let that address in over ssh (the rest of the plan stands). In `strict` the address must be inside a source the filter accepts for

> ssh: the address tacctl reaches the device from, at the head of `permit-mgmt`, a plain entry of the permit list, or an entry whose services cover tcp port 22. An entry with `udp/161` only does not count, and neither does the address the device authenticates against when the filter accepts only the server's answers from it (`permit-auth-replies`): `--from` with that address is refused, and the refusal says why. An apply makes the same check with the address it connects from.

---

5. Applying the change

The vendors differ in how a plan reaches the device. *Vendor Notes* compares them side by side.

5.1 Junos: let tacctl apply it

`apply` is the step after `plan` for a Junos device. It types the same statements into the device itself, under a confirmed commit, and proves the result, where `plan` prints them for a person to paste. A **rollout** is several devices changed in one go. Three words come up in it, each with a section of its own: a **shape** is one change as it applies to several devices (§5.2), a **canary** is the first device of a shape, applied alone before the rest of it (§5.3), and a **confirmed** shape is one that an earlier apply has already proved (§5.3).

A rollout in order. The devices are those of the scope `site-a`.

```
# 1. read the devices (apply reads each one again itself)
tacctl device config pull --scope site-a
# 2. read what is irregular on one device and what apply would type
tacctl device config plan rtr1
# 3. check every device, file the shapes, commit nothing
tacctl device config apply --scope site-a --check
# 4. one device: the review and the canary
tacctl device config apply rtr1
# 5. the rest, by hand with a question per device, or from a script
tacctl device config apply --scope site-a
tacctl device config apply --scope site-a --confirmed --yes
```

Step	What it does	Reads the device	Changes the device	Who
1. `pull`	Reads the devices and keeps what it read. `apply --pull` does this and the apply in one run	Yes	No	E (own scopes)
2. `plan`	Prints, in plain words, what is irregular on the device and the statements `apply` would type	No (uses the pull)	No: the statements are for a person to paste	E

Step	What it does	Reads the device	Changes the device	Who
3. `apply --check`	Loads the change in an exclusive configure session, runs the device's own `commit check`, discards the candidate. Each device that needs a change is named with its shape, filed as `new` unless it is already known	Yes	No: nothing is committed	E
4. `apply <name>`	Shows the change once per shape, asks for the device's name typed back, commits under `commit confirmed`, proves the result with a fresh login and confirms it. That is the review of the shape and its canary, and it turns the shape from `new` to `confirmed`	Yes	Yes: kept only when the proofs pass	E
5. `apply --scope`, or `--confirmed --yes`	Applies the rest. A device that is done reads `no change`. With `--scope` every shape of the run is reviewed and the first device of each goes alone (§5.3). `--confirmed --yes` is the form for a script: it applies the confirmed shapes only, asks nothing and skips the rest	Yes	Yes	E
Any time	`tacctl device config shapes` lists the shapes and where each device is	No	No	S

Only `apply` without `--check` writes to a device. An engineer sees the facts of a shape in the check lines and in the review (`new`, or `confirmed on <device>`); the `shapes` command is the superuser's.

Other forms of the command:

```
# read fresh, then apply only what differs now
tacctl device config apply --scope site-a --pull --differs
# also wait for you to test each class and type OK
tacctl device config apply rtr1 --hold
```

What happens during an apply:

- Every device is checked first. Then the changes are shown grouped by *shape* (the same change on several devices, §5.2), with secrets elided.
- You approve **each device by typing its name**. Anything else declines it.
- tacctl commits under `commit confirmed` (10 minutes by default). It then proves the change with a **fresh login as you** and checks what landed. Only after that does it confirm the commit. If the proof fails, it rolls the change back.
- A change of the Junos management filter is checked against the address the apply connects from, as `--from` is for a plan (§4.2). The review shows the plan's warnings as `warning:` lines, so the traffic a move to `strict` stops is listed before you approve.

- On a device with several members (a virtual chassis, two routing engines, a cluster), the commit check and the commits must answer for exactly the members `show version` lists. A member that is missing or answers otherwise fails the step, and the error names the member and its text. Over NETCONF, a reply of a virtual chassis or a cluster that names no member is refused; use the command-line transport, the default. A chassis of exactly one member (`show version` prints `fpc0:` even then) answers like a single device: its plain lines are read, and so is a block labelled with that one member; a block of another member, or a label twice, is refused.
- The master of a virtual chassis answers `commit check` and the commits without a label, and only the other members get a header (`fpc1:`). tacctl reads the lines no label opens as the answer of the one member of `show version` that has no block, and holds it to the same rules as the others: its success comes first, and on a commit its `commit complete` is the last line, after the other members' blocks. On a confirmed commit every member must print the note that such a commit prints, and a commit without it is refused. A line that is none of these, a label twice, and two members without a block fail the step.
- The first device of each shape goes alone (the canary, §5.3). By default at most 5 devices are changed per run (`--max`), with a pause between devices (`--pause`). §5.4 lists the options that shape a rollout.
- The time of the apply is not scaled where it is a timer on the device: the confirmed-commit window (10 minutes by default) is never multiplied by a timing scale. `--timing` shows where the time of each device went (§3).

> `PENDING until <time>` means tacctl could not read the device's state after the commit. **Type nothing on that device until the time passes.** The time is the device's own clock, in the zone the device prints. Any `commit` would confirm the change. The next apply or `--check` settles the record.

5.2 What a shape is

A shape is the same change on several devices. Two devices that need the same statements have one shape; two that need different ones have two. It is a 12-hex id: a hash of the statements of the change, as `plan` would type them, with two things taken out so that devices can be compared and the shape filed safely:

- every **secret** is removed (only its mark stays, so a shape never holds a value);
- every **per-device value** is masked, because it is the device's and not the change's: the SNMP location, contact and description, the host name, the login message and announcement, and the source address of a TACACS+ or RADIUS server. Two devices that differ only in these have the same shape. The order of the statements does not matter.

The shape comes from the plan made from the device's fresh read, so it is known at the check, before anything is loaded or committed. A device with no change has no shape. A change that would put a value into the shape has `no shape`: it is reviewed device by device, and `--confirmed` skips it.

A shape exists for two reasons. The review shows the change once per shape, not once per device: fifty switches that need the same change are usually one heading to read (devices of one shape

whose warnings, sections left out or kept statements differ get a heading each; each device is still asked for by name). And tacctl remembers what was proved: when a device took the change and passed the proofs, its shape is filed as confirmed, and a later run can apply it to other devices without a new review.

A worked example. Three devices of `site-a`: `rtr1` and `rtr2` need the same AAA change, and `jsw1` needs a different set of SNMP values.

```
$ tacctl device config apply rtr1,rtr2,jsw1 --check
Password for bob (device login):
  rtr1  checked: shape 3f2a1b9c0d4e (new) (aaa: 3 set, 1 delete); check ok
  rtr2  checked: shape 3f2a1b9c0d4e (new) (aaa: 3 set, 1 delete); check ok
  jsw1  checked: shape 8d41c7e2a5b0 (new) (snmp: 2 set); check ok

  3 checked
  Nothing was committed: --check ends after the device's own commit check.
```

There are two shapes: `3f2a1b9c0d4e` (`rtr1` and `rtr2`) and `8d41c7e2a5b0` (`jsw1`), both `new`. A review of this run has two headings, and reading `rtr1`'s statements covers `rtr2`'s. After `tacctl device config apply rtr1` has passed its proofs, the same check reads:

```
  rtr1  no change
  rtr2  checked: shape 3f2a1b9c0d4e (confirmed on rtr1) (aaa: 3 set, 1 delete); check ok
  jsw1  checked: shape 8d41c7e2a5b0 (new) (snmp: 2 set); check ok
```

`rtr1` is done, `rtr2` needs the same change as the one that worked, and `jsw1` needs a change nobody has proved yet.

5.3 Canary and confirmed

These are two different things. A **canary** is about the order inside one run; **confirmed** is about what an earlier run proved.

Canary. In a run, the first device of every shape is applied alone, and the rest of its shape start only when it applied. If it did not (it failed, was rolled back or is pending), the rest of that shape is skipped with `skipped: canary <name> failed`. At the default `--max-failures` of 1 that failure also stops the run, and every device not yet started, of any shape, is `not started` (failure limit reached); the other shapes go on only under a higher `--max-failures`. The canary is the first approved device of the shape in registry order, and every shape has one, `new` or `confirmed`. `--no-canary` starts the devices of a shape together.

With the default of one device at a time, the order of a run is the first device of each shape, one after the other, then the remaining devices in registry order, with `--pause` seconds from one device's end to the next one's start. For `rtr1`, `rtr2` and `jsw1` above that is `rtr1`, `jsw1`, `rtr2`: `jsw1` is the canary of its shape, so it goes before `rtr2`, which waits for `rtr1`.

Confirmed. A shape is in one of three states, shown in the `STATE` column of `tacctl device config shapes`:

State	How a shape gets there	What it allows
`new`	A check or an apply found the change on a device (`--check` files it, unless the shape is already known). Nothing has been proved	It can be applied after a review (the heading says `not confirmed yet`). `--confirmed` skips it: `unknown shape <hash>: review and apply it by hand first`
`confirmed`	An apply on one device went through: committed, a fresh login as you worked, what landed is the change the check showed, the planned sections read back whole, and the confirming `commit` was typed. The device is filed with the shape	`--confirmed` applies the shape to other devices, and with `--yes` asks nothing (the heading says `confirmed on <device>`)
`seeded`	A superuser ran `tacctl device config shapes seed <name>` for a device that was applied and verified by hand, and answered the question (`--yes` answers it). It is their word, logged as `device config shapes seed`	The same as `confirmed`

An apply that ends as applied by someone else's commit does not confirm its shape.

`shapes list` (superuser) has more columns than the state. `CONFIRMED` counts the devices that confirmed the shape. `NOW` counts the devices whose last check or apply is at it (from the records: no device is contacted), and their names follow the table, wrapped at 80 columns. `LINES` is the size of the shape. For a `new` shape the `NOTE` column says `nearest confirmed <hash>, N lines differ`: the confirmed (or seeded) shape whose statements differ least from this one, N being the lines that are in one of the two and not in the other, the `[section]` headings included. A small N means the change is a known one plus or minus a few statements; compare the two with `shapes list --show <hash>`. With no confirmed shape the note is `-`.

```
  SHAPE         STATE      CONFIRMED  NOW        LINES  NOTE
  3f2a1b9c0d4e  confirmed  1 device   2 devices  5      -
  8d41c7e2a5b0  new        -          1 device   3      nearest confirmed 3f2a1b9c0d4e,
                                                          8 lines differ

At a shape now (last check or apply):
  3f2a1b9c0d4e  rtr1, rtr2
  8d41c7e2a5b0  jsw1
```

(The title of the table and the rules under it are left out here, and the note is wrapped; a shape one line away reads `1 line differs`.) `--confirmed --yes` is the form for a script: it applies the confirmed shapes and skips every other device, `jsw1` here, with the row `skipped: unknown shape 8d41c7e2a5b0: review and apply it by hand first`. `--confirmed` without `--yes` selects the same devices but still asks for each by name, so it needs a terminal.

5.4 The options of a rollout

Option	What it controls	Default and range
`--check`	Stop after the device's own `commit check`: nothing is committed, the device's shape and result are recorded. `--exit-code` with it exits 2 when a device needs a change. Not with `--hold`, `--pause` or `--no-canary` (exit 2)	
`--confirmed`	Which shapes are applied: only the confirmed and seeded ones, the other devices are skipped. The one form that `--yes` answers the review of	
`--yes`	Answers every device's question (logged as `mode=scripted`). Only with `--confirmed` (with `--check` it does nothing)	
`--no-canary`	Starts the devices of a shape together, not the first alone	The first alone
`--max <n>`	How many devices one run applies, counted after `--confirmed`, in registry order; the rest are `skipped: beyond --max <n>: run it again for the rest`. The check still covers every selected device	5; 1 to 1000
`--pause <seconds>`	The wait from one device's end to the next one's start. The first device is not delayed; the wait also separates a canary from the rest	The `apply-pause` setting, 60; 0 to 3600
`--concurrency <n>`	How many devices are worked on at a time. Never above `device.config.max_concurrency`; `--hold` takes only 1	Apply 1; check: the setting (8)
`--max-failures <n>`	After n failed devices no new one is started (the running ones finish). A failed canary counts. One refused password stops the run at once, in either phase	Apply 1; check none
`--hold`	After the fresh login, keep the change only when you log in with one account of each class and type `OK`	

5.5 Cisco: paste the plan

1. If you asked for a restore point, paste that block first, on its own, at the privileged prompt. Then check that `dir flash:` lists the file.
2. Keep a **second session** logged in.
3. Paste the region from `PASTE FROM HERE` to `PASTE TO HERE`.
4. Verify as the plan says. Then open a **new** session with your own TACACS+ account and run `show privilege`.

Command authorization is held back on purpose. Until the device is proven to use the scope's server, the plan leaves out `aaa authorization commands` and `config-commands`. Once a commands list names the server, any user the server does not know is denied every command. Pull again after the first paste, and the next plan adds them.

`enable` always uses the **local** enable secret. Users land at their level directly (superuser 15, operator 7), so nobody needs `enable` day to day.

5.6 WTI: type the checklist

A WTI unit has no configuration to paste. Its plan is a numbered checklist of menu keystrokes (`/N …`) plus one IP-tables block. Before you start:

- check `/H` (firmware) and `/SN` (other admins logged in);
- keep a serial console or a second session open.

Values take effect as soon as you enter them, and there is no rollback. The plan never types the local accounts or the final IP-tables `DROP`.

---

6. Verify, then clean up

```
tacctl device config pull sw1 --diff      # should read ok for the sections you changed
tacctl ssh sw1                            # a new login as yourself works
tacctl device config plan sw1 --converge  # optional: remove old servers and similar extras
```

---

7. When it goes wrong

Situation	Action
Junos apply failed its proof	Nothing to do: tacctl already rolled it back. Read the row's reason.
Junos device `PENDING`	Wait out the window. Do not type `commit`. Run `apply --check` afterwards.
Cisco paste broke something, session still open	Paste the back-out: `tacctl device config plan <name> --backout`. With a restore point, use `configure replace` with the restore-point file.
Locked out of the device entirely	Ask a **superuser** for `tacctl breakglass hold <name> --reason "..."`. Log in with the scope's local break-glass account, fix the device, then the superuser releases the hold (*Break Glass*).
Host key mismatch on pull or ssh	Check the key on the console first, then `tacctl device hostkey <name> set SHA256:<fp>` (*Connecting to Devices* §5)
`failed: calls itself '<x>'`	The device's own hostname doesn't match the registry. Check you have the right box, then fix the registry or the device.
`failed: other editors or staged changes`	Someone else has uncommitted configuration on the device. Coordinate with them.

Situation	Action
Apply fails and names a member (`fpc1`, `re1`, `node1`)	That member of the virtual chassis, routing-engine pair or cluster was missing from the answer of the check or the commit, or answered with an error, a warning or no text. Read its text in the error and look at the member on the device.
Plan refuses the Junos filter: `... is not inside a source of the rendered strict filter that is accepted for ssh`	The strict filter would cut you off. Ask a superuser to add your address to the permit list the scope renders from (plain, or with `tcp/22`), then plan again (§4.2)

---

8. Quick reference

Task	Command
Register + read	`tacctl device add <n> <ip> --vendor <v> --pull`
Enroll discovered	`tacctl device enroll --discovered`
Read a scope	`tacctl device config pull --scope <s>`
Where the time of a run goes	add `--timing` to `pull`, `diff --pull`, `plan --pull` or `apply`
Scale in force for a slow device	`tacctl device show <n>` (`Timeout scale` row)
What differs	`tacctl device config list --differs`
Line-level diff	`tacctl device config diff <n>`
Paste plan	`tacctl device config plan <n>`
Junos dry check	`tacctl device config apply <n> --check`
Junos apply	`tacctl device config apply <n>`
Junos rollout of a scope, in order	`apply --scope <s> --check`, then `apply <n>`, then `apply --scope <s>` (§5.1)
Junos rollout from a script	`tacctl device config apply --scope <s> --confirmed --yes`
Verify	`tacctl device config pull <n> --diff`
Back-out	`tacctl device config plan <n> --backout`
Walkthrough for a device	`tacctl device config show <n>`
Register a copy at a bench address	`tacctl device clone <n> <new> --staging <ip>`
Move the copy into place	`tacctl device rename <new> <n> --replace`
Test the AAA source addresses	`tacctl device check <n> --sources`

Task	Command
Junos filter mode of a scope	`tacctl scope mgmt-acl <s> mode`
Re-pin a key	`tacctl device hostkey <n> set SHA256:<fp>`

See also: Vendor Notes · Day-to-Day Device Operations · Troubleshooting §4 · *Break Glass* · `man tacctl`